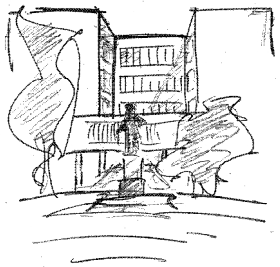
[P273]
Пројектовање база података

11

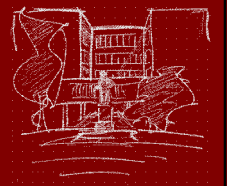


Саша Малков

Универзитет у Београду
Математички факултет

2023/2024

[P273]
Пројектовање база података



Саша Малков

Тема 11

Репликација

Дистрибуирани системи и поузданост

- У дистрибуираним системима је реалност да неке од дистрибуираних компоненти нису функционалне
  - Није питање “да ли” него само “када, колико и којих” компоненти ће постати неисправно у неком тренутку
- Неисправности обухватају:
  - багове
  - људске грешке при управљању системима
  - преоптерећеност (оперативна)
  - загушење (комуникационо)
  - оштећења медија за чување података
  - оштећења електронских уређаја
  - злонамерне активности

Однос према насталим неисправностима

- Изоловање неисправности
- Толеранција неисправности

Изоловање неисправности

- Принцип:
  - Обезбеђивање да неисправност једне компоненте не утиче на функционалност осталих компоненти
- Последице:
  - Функција коју је обезбеђивала неисправна компонента није расположива док се проблеми не отклоне и компонента не постане оперативна
  - Остале компоненте функционишу у мери у којој то могу без неисправне компоненте

Толеранција неисправности

- Принцип:
  - У случају неисправности неке од компоненти, друге компоненте преузимају њене функције
- Последице:
  - Повећана флексибилност и поузданост система
  - Повећана сложеност и цена система

Остваривање толеранције

- Основни методи:
  - Реконфигурисање система
  - Прикривање неисправних компоненти
- Заједничке особине:
  - Неопходна је редунданднost
  - Корисници система не опажају неисправности
    - осим евентуално кроз умањене перформансе система

Аспекти поузданости

- Поузданост система (*dependability*) чине
  - Расположивост система (*availability*)
    - “способност система да прими захтев”
    - релативно ниско трајање периода када систем “не ради”
    - или ретко долази до неисправности или се оне брзо отклањају
  - Одговорност система (*reliability*)
    - “способност система да одговори на захтев”
    - систем или уопште неће имати неисправности током рада или ће оне бити превазилажене у ходу тако да не буде прекида оперативности
    - или ретко долази до неисправности или је систем у потпуности способан да се довољно брзо опорави да оне не ометају рад

Расположивост и одговорност

- Систем може бити високо одговоран а да није високо расположив
  - Пример: систем који не прима захтеве током прављења резервних копија, а у међувремену ради са ретким неуспесима
- Систем може бити високо расположив а да није високо одговоран
  - Пример: систем који је скоро увек у стању да прими захтев (нпр. аутоматски се превазилазе евентуални кварови) али релативно често не успева да одговори на захтев (нпр. због неисправности софтвера)

Мера одговорности

- Систем је потпуно одговоран ако никада не трпи оштећења током обраде захтева
- Мера одговорности је вероватноћа да ће систем одговорити на захтеве које је примио
  - $R(t) = 1\text{-}F(t)$
    - $R(t)$ је одговорност у интервалу времена дужине t
    - $F(t)$ је вероватноћа да ће доћи до грешке у интервалу дужине t
  - $R(t) = 1 - \int_0^t f(x)dx$
    - $f(t)$ је густина вероватноће неуспеха

Мера одговорности (2)

- Уобичајена мера је “средње време до неуспеха” (*mean time to failure*), у ознаци MTTF:
  - $MTTF = R^{-1}(0.5)$
  - т.j. време за које је $R(t) = 0,5$
- Користи се и мера “средње време између неуспеха” (*mean time between failures*), у ознаци MTBF

Мера расположивости

- Систем је потпуно расположив ако је увек у стању да прими захтев
- Мера расположивости $A(t)$ је вероватноћа да ће систем бити у стању да прими захтев у неком тренутку t
  - т.j. да ће систем
    - или радити исправно у интервалу $[0,t]$
    - или ће последњи проблем бити отклоњен у неком тренутку x, где је $0<x<t$
- Ова мера се назива и “тренутна расположивост”

Мера расположивости (2)

- Уобичајена мера је и “средње време оправке” (*mean time to repair*), у ознаци MTTR:
  - Очекивано трајање поправљања система након неуспеха
  - Код база података, назива се и “средње трајање опоравка”

Мера расположивости (3)

- У пракси се често употребљава тзв. гранична расположивост:

$$\lim_{t\to\infty} A(t) = \frac{MTTF}{MTTF + MTTR}$$

- Поред тога, употребљава се и метод експерименталног мерења:
  - Ако се у интервалу $[0,t]$ региструју интервали u_i у којима систем није расположив, онда је расположивост:

$$A(t) = 1 - \frac{\sum_i u_i}{t}$$

Мера поузданости

- Нека је s сервис чије извршавање траје τ
  - ***Расположивост*** $A_s(t)$ је вероватноћа догађаја $I_s(t)$ да ће сервис бити успешно иницран у тренутку t
  - ***Одговорност*** $R_s(t, \tau)$ је условна вероватноћа догађаја $T_s(\tau)$ да ће систем успешно да заврши обраду захтева, под условом да је наступио догађај $I_s(t)$ (тј. да је захтев примљен)
  - ***Поузданост*** система је вероватноћа $D_s(t, \tau)$, која представља конјункцију догађаја $I_s(t)$ и $T_s(\tau)$, тј. да ће захтев бити успешно и примљен и обрађен

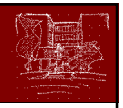
Мера поузданости (2)

- $A_s(t) = P[\, I_s(t) \,]$
- $R_s(t, \tau) = P[\, T_s(\tau) \mid I_s(t) \,]$
- $D_s(t, \tau) = P[\, I_s(t) \wedge T_s(\tau) \,] = A_s(t)\, R_s(t, \tau)$

Шта је репликација?

- Репликација је свако удвајање неке компоненте рачунарског система које доноси неки ниво редундантности

Зашто реплицирамо?



- Два основна мотива:
  - Перформансе
  - Поузданост
- У случају дистрибуираних база података репликација је често неопходно средство за достизање потребног нивоа и перформанси и поузданости
  - Штавише, чест је и обрнут случај, да се прави дистрибуирана база података само да би се остварила репликација

Подизање перформанси

- Остварује се кроз расподелу оптерећења на реплике
- Посебно је значајно у случају читања
- Може да има негативан утицај на перформансе писања
  - посебно ако се очекује да све реплике изврше ажурирања у току трансакције
  - решава се одложеним реплицирањем
    - тако пада поузданост, па често није прихватљиво
    - али писање може чак и да буде брже него без реплика

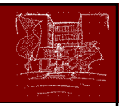
Подизање поузданости

- Имплементација толеранције неисправности
  - кроз маскирање неисправности
  - и реконфигурисање система
- Препознавање неисправности
  - кроз “дуплирање” (*mirroring*)
    - дуплирање је најједноставнији облик репликације

Шта реплицирамо

- Предмет репликације могу бити:
  - подаци
  - процеси
  - објекти
  - поруке
- Нас првенствено занима репликација **података**

Предмет репликације података



- База података
- Табела
- Фрагмент табеле
- Глобални каталог базе података
- Систем датотека
- Појединачна датотека

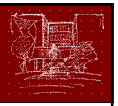
Ограничења репликације

- Основну цену репликације представљају
  - додатна цена простора због редундантног чувања података
  - повећана цена писања због мењања података на више локација
  - повећан обим преноса података ради репликације
    - објективно се умањује за уштеде због приступања ближим подацима
  - цена додатне обраде услед имплементације репликације
- Ограничење ефикасности
  - ако је ограничена расположивост једне копије A, онда је ограничена и расположивост реплицираног система
    - тачно ограничење зависи од модела репликације и врсте употребе
    - нпр. расположивост ажурирања локације не може да пређе $A^{1/2}$

Начини репликације података

- Основни принцип
  - Подаци се одржавају на више (дистрибуираних) локација
- Основни начини репликације
  - Читај један пиши све (*ROWA*)
  - Консензус кворума (КК) (гласање)
  - Консензус кворума у уређеним мрежама

Протоколи *ROWA*



- "Читај један пиши све"
- "*Read One Write All*"

- Концепт
  - Само примарна копија се чита
  - Све копије се мењају

Протоколи *ROWA* (2)

- Протоколи *ROWA*
  - толеришу отказе локација када је у питању читање
  - не толеришу отказе локација када је у питању писање
  - не толеришу отказе комуникација

- Варијанте
  - Основни *ROWA* протокол
  - *ROWA-A*, са мењањем расположивих копија
  - *ROWA* са примарном копијом
  - *ROWA* са токенима правих копија

Основни *ROWA* протокол

- Протокол *ROWA*
  - преводи сваку операцију читања у једну операцију читања, на једној од копија
  - преводи сваку операцију писања у *N* операција писања, по једну на свакој копији

- Контролер конкурентности на свакој локацији синхронизује приступ копијама
  - свака промена мора да успе на свим локацијама или ни на једној

- У случају отказивања неке локације
  - читање је и даље могуће
  - писање није могуће до опоравка неисправне локације

Протокол *ROWA* – A

- "*Read One Write All Available*"
- Разлика у односу на основни протокол:
  - писање се изводи не на "свим" него на "свим расположивим" копијама
- У случају отказивања неке локације
  - читање је и даље могуће
  - писање је и даље могуће
  - неисправне локације могу да постану расположиве тек након опоравка уз пуно преписивање свих измена насталих током неоперативног периода

Протокол *ROWA* – А (2)

- "Алгоритам расположивих копија"
  - синхронизује неуспехе и опоравке контролисањем расположивости копија
- Операције писања се шаљу свим копијама:
  - ако нека локација није оперативна, од ње нема одзива у допуштеним оквирима (*missing write*)
  - ако јесте, онда је операција писања обрађена или одбачена
- Могућа су унапређења ради откривања грешака које се односе на препознавање статуса учесника

Протокол *ROWA* – А (3)

- Пре потврђивања трансакције координатор започиње примену двофазног протокола валидације:
  - "валидација пропуштених писања" - проверава да ли су све копије које су пропустиле неко писање још увек неоперативне
    - координатор шаље поруку *UNAVAIL* свим копијама од којих још није добијен одговор на захтев за писање
    - ако је нека постала активна, и покренула писање, она ће примити овај захтев и одговорити сигналом за прекид трансакције
    - ако није ниједна постала активна, наставља се са следећим кораком
  - "валидација приступа" - проверава да ли су све копије које су читале или писале још увек расположиве
    - координатор таквим копијама шаље поруку *AVAIL*
    - свака која је активна прима поруку и потврђује да је активна
    - ако нека није активна, прекида се трансакција

Протокол *ROWA* – А (4)

- Проблеми:
  - Свакој неисправној локацији се шаљу бар два захтева (1. операција писања, 2. провера у току валидације) и на сваки се чека највише допуштено време
    - тиме се потенцијално губи значајно време и смањује одзивност система
  - Иако безбедан, поступак двофазне валидације није ефикасан
    - пожељно је препознати опоравак локација у што краћем интервалу
    - овај поступак игнорише опоравак у неким случајевима и одлаже га за касније, чиме му се потенцијално подиже цена

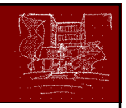
ROWA са примарном копијом

- Разлика у односу на основни протокол:
  - једна копија се проглашава за "примарну"
  - остале копије су "резервне"
  - читање се изводи само са примарне копије
  - писање се изводи на примарној и свим расположивим резервним копијама
- Овај протокол
  - не подиже перформансе читања
  - унапређује само поузданост система

ROWA са примарном копијом (2)

- Ако примарна копија постане неоперативна, изабрана резервна копија (по унапред одређеној линији преузимања) постаје нова примарна
  - да би ово било могуће потребно је да буде могуће недвосмислено разликовати неоперативност примарне копије од проблема у комуникацији
  - у супротном се може догодити да постоји више примарних копија, што је фатално по конзистентност
- Ако резервна копија постане неоперативна
  - она не може постати примарна све док не буде у потпуности опорављена
  - може преузети неке улоге резервне и након делимичног опоравка

ROWA са примарном копијом (3)



- Иако је овај алгоритам релативно једноставан и има неких слабости, ипак је један од најчешће употребљаваних
  - наравно, само у случајевима где је једини циљ подизање нивоа позданости

ROWA са токенима "правих" копија

- Разлика у односу на основни протокол:
  - сваки податак, у било ком тренутку времена, има или један *ексклузиван* токен или скуп *дељивих* токена
  - операција писања мора да добије ексклузиван токен
  - операција читања мора да добије бар дељиви токен
- Концептуално слично дистрибуираном закључавању
- "Узимање" токена обухвата, по потреби, и ажурирање податка

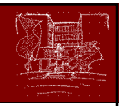
ROWA са токенима "правих" копија (2)

- При писању:
  - када копија мора да изврши операцију писања, она лоцира и узима ексклузиван токен за конкретан податак
  - ако он не постоји, она лоцира и проглашава за неисправне све дељиве токене за конкретан податак и прави нови ексклузиван уместо њих
- При читању
  - када копија мора да изврши операцију читања, копија лоцира дељиви токен, копира његову вредност и прави и чува нови дељиви токен
  - ако не постоји дељиви токен, она лоцира ексклузиван токен и конвертује га у дељиви и затим поступа као у претходном случају

ROWA са токенима "правих" копија (3)

- Токен има обједињену семантику налик на “локални катанац” на “дистрибуираном податку”
- Представља механизам за препознавање конфликата између више писања или између писања и читања података
- У случају проблема
  - само локације које имају токен на неком податку, могу да користе тај податак
  - ако су сви токени на податку лоцирани на неоперативној локацији, онда тај податак није расположив
  - при активирању неке локације A
    - сви њени ексклузивни токени се задржавају
    - за сваки податак, за који на лок.A постоји дељиви токен, се проверава да ли постоји и на некој од оперативних лок., и ако постоји онда се поништава на лок.A
- Постоје и другачији алгоритми који почивају на токенима, који се везују, на пример, за поруке и комуникацију...

ROWA са токенима "правих" копија (4)



- Након писања промена би требало да се дистрибуира
- То може да се остварује различитим механизмима
- На пример, на сваком од чворова асинхрони процес обилази све ексклузивне токене, конвертује их у дељиве и копира измењене податке (и дељиве токене) на (одређен број) других локација...

Консензус кворума (гласање)

- За разлику од метода *ROWA*
  - допушта писање на подскупу (*кворум писања*) оперативних локација
  - читања се изводе са поскупа (*кворум читања*) локација који мора да има непразан пресек са кворумом писања
- Правило *пресека кворума* обезбеђује да ће свако читање моћи да се одвија са најсвежије писаним вредностима
- За локацију која учествује у операцији се каже да је *гласала* за операцију

Консензус кворума (гласање) (2)

- Ова техника маскира неисправности, без додатних захтева (поступака) при опоравку неисправних локација
  - ефикасније него у случају валидације у два корака
- Кворуми могу бити
  - статички – одређени гласањем при подизању система
  - динамички – ако локације могу да се реконфигуришу

Консензус кворума (гласање) (3)

- Врсте консензуса кворума:
  - КК са униформном већином
  - КК са тежинском већином
  - КК са тежинском већином за директоријуме
  - Уопштени КК за апстрактне типове података
  - Хибрид *ROWA* / КК

КК са униформном већином

- Најстарији од метода кворума
- Операција читања или писања успева акко већина локација "одобри" извршавање
- Не морају све локације које су гласале и да изврше операцију на својим копијама података
  - Читање је довољно да се изврши на копији која има најсвежију вредност
    - иако се припрема (провера конкурентности и провера верзије података) одвијају на свим копијама кворума
  - Писање мора да се изврши на свим копијама кворума
- Почива на препознавању евентуалних конфликата од стране управљача конкурентношћу на појединачним локацијама

КК са униформном већином (2)

- Постиже се отпорност на неисправности
  - на локацијама
  - у комуникацији
    - (све док постоји већинска група локација које могу да међусобно комуницирају)
- Цена
  - релативно висока и при читању и при писању
  - бар половина +1 локација мора учествовати у свакој операцији

КК са тежинском већином

- Представља уопштење алгоритма КК са униформном већином:
  - Свака копија d_s податка d добија не-негативну *тежину* тако да сума свих тежина на једном податку буде u.
  - Сам податак d добија праг читања r и праг писања w, тако да важе наредни услови:
    - $r + w > u$
      - еквивалентно непразном пресеку кворума
    - $w > u/2$
      - еквивалентно већини
      - неопходан ако се најсвежија копија установљава на основу броја верзије
      - није потребан ако се користе временски печати (видећемо касније)
  - Кворум читања (или писања) податка d је сваки скуп копија чија је тежина бар r (или w)
  - Тежине могу да се разликују за различите податке

КК са тежинском већином (2)

- За установљавање актуелности копија се обично употребљава механизам верзија података
  - Свака копија податка је означена бројем верзије, који се иницијално поставља на 0
  - При свакој промени се уписује нови број верзије, који је за 1 већи од највећег у кворуму писања
  - При читању са бира вредност копије са највећим бројем верзије
- Алтернатива је употреба временских печата
  - Свака копија податка је означена временом записивања
  - При свакој промени се уписује време промене
  - При читању са бира вредност копије са највећим временом промене

КК са тежинском већином (3)

- Свака операција писања $w[d]$ се преводи у скуп појединачних операција писања $w[d_s]$ на свим копијама у неком кворуму писања:
  - читање свих копија у кворуму
  - читање њихових верзија
  - повећавање максималног броја верзије
    - *newVersion = max(version) + 1*
  - писање свих копија са новим бројем верзије

КК са тежинском већином (4)

- Свака операција читања $r[d]$ се преводи у скуп појединачних операција читања $r[d_s]$ на свим копијама у неком кворуму читања:
  - читање свих копија
  - читање њихових верзија
  - копија са највећим бројем верзије (*текућа копија*) представља резултат

КК са тежинском већином (5)

- Због већинског гласања, у сваком кворуму читања сигурно постоји текућа копија података
- Овај метод може да сарађује са сваким алгоритмом за контролу конкурентности који омогућава серијализујуће извршавање

КК са тежинском већином (6)

- Начелно се не захтева *сложен* алгоритам опоравка
  - Копија која је пропустила неко ажурирање неће имати последњу верзију податка, па она неће бити коришћена (враћена као резултат) бар до првог наредног ажурирања
- У пракси може да буде пожељно да се у случају отказа промене прагови кворума
  - пре тога мора да се обезбеди ажурирање довољног броја активних копија
  - иначе може да се догоди да неки смањен кворум читања нема ажурну вредност, па да читање буде неисправно
  - зато је први корак вид опоравка

КК са тежинском већином (7)

- Алгоритам је флексибилан
  - Мењањем параметара r и w и додељивањем тежина појединачним копијама појединачних података се:
    - управља прављењем кворума и
    - прилагођава различитим поузданостима компоненти
  - у случају екстремних вредности параметара r и w алгоритам се своди на *ROWA* (r=u, w<=min(d_s))
  - у случају уједначених вредности свих параметара d_s, као и једнаких вредности r и w, алгоритам се своди на КК са униформном већином

КК са тежинском већином (8)

- Избор тежина и прагова може да буде сложен посао
- Алтернатива је аутоматско одређивање тежина
- Један начин је поступком *каљења*
  - у општем случају, каљење је поступак постепеног (или поновљеног) *хлађења*, ради достизања минималног *енергијског стања*
  - у случају КК, тежи се достизању максималног нивоа расположивости
  - исти значај имају кворуми читања и писања
  - циљ је да се тежине локација одређују пропорционално поузданости
  - каљење се имплементира тако да се изабраном пару тежина (v_i,v_j) додаје неки од парова (-1,-1), (-1,1), (1,-1), (1,1)
  - начин избора парова и додељене вредности зависи од варијанте алгоритма

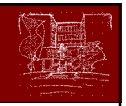
КК са тежинском већином за директоријуме

- Уопштење за *директоријуме*
  - Директоријум је пресликавање *простора кључева* у *простор вредности*
  - Примењује се и на нерелационе базе засноване на каталозима
- Постоје следеће операције на директоријумима:
  - ***Insert*(*Key k*, *Value v*)** – додељивање вредности v кључу k, само ако кључ k не постоји у директоријуму
  - ***Update*(*Key k*, *Value v*)** – додељивање нове вредности v кључу k, само ако кључ k постоји у директоријуму
  - ***Delete*(*Key k*)** – брисање кључа k из директоријума, само ако кључ k постоји у директоријуму
  - ***Value Lookup*(*Key k*)** – читање вредности додељене кључу k (или false, ако не постоји)

КК са тежинском већином за директоријуме (2)

- Мења се грануларност операција тако да се не извршавају на “великим” директоријумима него на њиховим мањим деловима

Уопштени КК за апстрактне типове података



- Сличан проблем као за директоријуме
- …

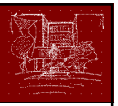
Хибрид *ROWA* / КК

- Основна слабост метода КК је неоправдано висока цена у случају ретких отказивања комуникација
- Хибридни приступ редукује цену кроз
  - примену метода *ROWA* током поузданих оперативних периода
    - тзв. “*нормални режим*”
  - примену метода КК у условима постојања отказа локација или комуникација
    - тзв. “*режим отказа*”
  - трансакције могу да започну рад у нормалном режиму и да током рада пређу у режим отказа

Хибрид *ROWA* / КК (2)

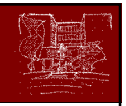
- Постојање отказа се препознаје кроз *пропуштена писања*
  - Ако трансакција препозна да је било изостајања писања неке копије податка коју је већ читала, она мора да буде прекинута
  - Ако трансакција препозна да је било изостајања писања неке копије пре читања са те копије, онда мора да пређе у режим КК и да чита са копија које нису пропустиле писање
    - кворуми сигурно обухвата бар једну локацију која није пропустила писање

Хибрид *ROWA* / КК (3)



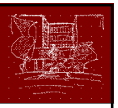
- Ако трансакција *T* упути захтев за писање и дође до изостајања писања на некој копији
  - сама трансакција *T* то лако уочава и мења режим
  - да би друге трансакције то дознале потребни су додатни механизми
- Један начин је вођење листе пропуштених писања (*MWL*) уз сваку копију податка

Консензус кворума у уређеним мрежама



- Савремени алгоритми репликације поред толерисања неисправности стављају у први план и оптимизацију трошкова
- Покушава се постизање смањивања броја локација које учествују у гласању увођењем логичке структуре у скуп локација

Консензус кворума у уређеним мрежама (2)



- Врсте:
  - Алгоритам √n
  - Протокол матрице (*GRID*)
  - Асимптотска висока расположивост
  - Дрво кворума
  - КК са хијерархијском тежинском већином
  - КК са вишедимензионалном тежинском већином

Алгоритам √n

- Уводи се претпоставка да неће бити отказивања комуникације
- Свакој локацији се додељује један кворум који има непразан пресек са свим кворумима који су додељени другим локацијама
  - свака два кворума имају непразан пресек
- Трансакција мора да обезбеди консензус свих локација у кворуму додељеном његовој матичној локацији

Алгоритам √n (2)

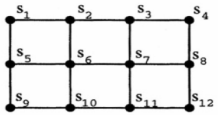
- Када локација s_i затражи међусобно искључивање, протокол тражи консензус од кворума локација Q_i тако да важи:
  - $Q_i \cap Q_j \neq \varnothing$ – за свака два кворума мора постојати бар једна заједничка локација, која служи као арбитар када се неки чланови двају кворума не слажу
  - $s_i \in Q_j$ – локација која поставља захтев то чини у односу на себе, без додатних порука
  - $|Q_i| = K$ – свака локација шаље и прима исти број порука
  - $|\{Q_i : s_i \in Q_i\}| = M$ – свака локација је арбитар за исти број локација, тј. учествује у истом броју кворума

Алгоритам √n (3)

- Претходни услови имплицирају
  $n = K\,(K\text{-}1) + 1$
  - са K размена порука се може договорити n локација
- Протокол постиже узајамно искључивање са $c\sqrt{n}$ порука $(3 \leq c \leq 5)$

Протокол матрице (GRID)

- Скупови локација се уређују у облику матрице са N колона и M врста. тако да је $M^*N \leq n$:



- Један кворум читања обухвата локације са по тачно једном локацијом из сваке колоне
- Кворум писања се састоји од свих локација једног кворума читања и једне колоне

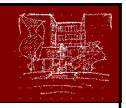
Протокол матрице (GRID) (2)

- Основни циљеви такве организације кворума су
  - пресек сваког кворума писања и сваког кворума читања
    - свако писање успева на свим чворовима једне колоне
    - свако читање обухвата бар један чвор те колоне и добија свеже податке
    - поуздана имплементација механизама закључавања
  - равномерно распоређивање оптерећења
  - пресек између свака два кворума писања
    - поуздана имплементација механизама закључавања

Протокол матрице (GRID) (3)

- Скуп локација G је C-покривање ако свака колона има непразан пресек са G.
- Операција читања бира произвољну пермутацију π_r која се састоји од M локација у произвољном реду r
  - π_r одређује редослед комуницирања ради постављања катанаца за читање на C-покривачу
    - обезбеђује равномерно оптерећење
  - ако успе, гарантује се да једна од копија има најсвежију вредност
  - ако не успе, покушава се слично на наредној врсти док се не добије катанац
  - ако не успе ни на једној врсти, операција се прекида и катанци ослобађају

Протокол матрице (GRID) (4)



- Операција писања:
  - Најпре закључава C-покривач користећи протокол читања
  - Затим закључава све локације произвољне колоне c у редоследу одређеном пермутацијом π_c
  - Да би нека два писања радила конкурентно, свако од њих мора да закључа по C-покривач и једну колону, па се мора препознати конфликт
    - слично и за конфликт читања и писања

Литература за ову тему

- *A.A. Helal, A.A. Heddaya, B.B. Bhargava,* ***Replication Techniques in Distributed Systems****, Kluwer Academics Publishers, 1996.*